Equality — first guess

Two (probabilistic) programs are equal iff from all initial states they have equal probability of establishing equal postconditions: for example,

\[
\begin{array}{c|c}
\text{coin:= heads} & 2/3 \\
\text{coin:= tails} & 1/3 \\
\end{array}
\]

are equal.

What about

\[
\begin{array}{c|c}
\text{coin:= edge} & 0 \\
\text{coin:= heads} & 1/2 \\
\text{coin:= tails} & 1/2 \\
\text{edge, heads} & 1 \\
\text{edge, tails} & 1 \\
{\text{edge, heads, tails}} & 1 \\
\end{array}
\]

Non-deterministic choice.

\[
\begin{array}{c|c}
\text{coin:= edge} & 1 \\
\text{coin:= heads} & 1/2 \\
\text{coin:= tails} & 1/2 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{coin:= edge} & 1 \\
\text{coin:= tails} & 0 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{coin:= edge} & 1 \\
\text{coin:= heads} & 0 \\
\text{coin:= tails} & 0 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{coin:= edge} & 1 \\
\text{coin:= heads} & 0 \\
\text{coin:= tails} & 0 \\
\end{array}
\]

Are they equal?
Equality — first guess

Two (probabilistic) programs are equal iff from all initial states they have equal probability of establishing equal postconditions: for example,

\[ \text{coin} := \text{edge} \quad \text{and} \quad \text{coin} := \text{heads} \]

\[ 2/3 \quad \text{and} \quad 1/2 \]

are equal.

What about

\[ \text{coin} := \text{edge} \quad \text{and} \quad \text{coin} := \text{tails} \]

\[ 1/2 \quad \text{and} \quad 1/2 \]

Are they equal? Apparently they are.

Equality — first guess, wrong guess

But **should** they be?

No, they should not — and their indistinguishability in this simple probabilistic logic makes it **non-compositional**.

The logic we now present represents the “least extra effort” — in a sense that can be made precise — that regains compositionality.

What about

\[ \text{coin} := \text{edge} \quad \text{and} \quad \text{coin} := \text{heads} \]

\[ 1/2 \quad \text{and} \quad 1/2 \]

\[ (\text{coin} := \text{edge} \quad \text{and} \quad \text{coin} := \text{tails}) \]

Are they equal? Apparently they are.
Probabilistic-program logic: introduction

What is the probability that the probabilistic program

\[ \text{coin} := \text{heads} \quad \frac{1}{2} \quad \text{coin} := \text{tails} \]

establishes the postcondition \( \text{coin} = \text{heads} \)?

We can abbreviate "\( \text{coin} := \text{heads} \quad \frac{1}{2} \quad \text{coin} := \text{tails} \)" as just

\[ \text{coin} := \text{heads} \quad \frac{1}{2} \quad \text{tails} \]

because the left-hand sides "\( \text{coin} := \)" are the same.

Probabilistic programs

1. Assignment statements;
2. Probabilistic choice;
3. Conditionals;
4. Sequential composition;
5. Demonic choice.

We will look at these in turn; what we need to know for each type of program fragment \( \text{prog} \) is

What is \( \text{wp.} \text{prog} \) \( B \) for arbitrary postcondition \( B \)?

The usual technique for setting this out is structural induction over the syntax of the programming language.

Probabilistic programs: assignment statements

\[ x := E \]

Assign the value of expression \( E \) to the variable \( x \).

\[ \text{wp.} \quad (x := x+1) \]

\[ B \quad \text{wp.} \quad (x := x+1) \]

Definition.

\[ x := E \]

Informal description.

\[ x := x+1 \]

Syntactic substitution.

Example.


Probabilistic programs: embedded Booleans

wp.(x:= x+1).\{x=3\}
= \{x=3\}\{x:= x+1\}
definition
= \{(x+1)=3\}
substitution
= \{x=2\}
arithmetic

The probability that x:= x+1 achieves x=3 is one if x=2 initially, and zero otherwise.

Thus "\[\]" must be an embedding function that takes true to one and false to zero.

Probabilistic programs: syntactic “sugar”

if G then prog fi
If guard G holds, then execute the body prog; otherwise do nothing.

if G then prog else skip fi
Do nothing.

skip

if G then prog1 else prog2 fi
If guard G holds, then execute prog1; otherwise execute prog2.

Probabilistic programs: probabilistic choice

\[\text{wp}(\text{prog}_1 p \oplus \text{prog}_2)B = p \times \text{wp}\text{.prog}_1B + (1-p) \times \text{wp}\text{.prog}_2B\]

\[\text{wp}(c:= H 1/2 @ T),\{c=H\}\]
= \(1/2 \times \text{wp}(c:= H),\{c=H\}\)
+ \((1-1/2) \times \text{wp}(c:= T),\{c=H\}\)
edefinition
= \(1/2 \times [H=H] + 1/2 \times [T=H]\)
assignment
= \(1/2 \times 1 + 1/2 \times 0\)
embedding
= \(1/2\).
arithmetic

Probabilistic programs: conditional

\[\text{wp}(\text{if } x \geq 1 \text{ then } x:= x - 1 \text{ else } x:= x + 2 \text{ fi},\{x \geq 2\}\]
= \(\text{wp}(x:= x - 1),\{x \geq 1\}\times \text{wp}(x:= x + 2),\{x \geq 2\}\)
"sugar"
= \([x \geq 1] \times \text{wp}(x:= x - 1),\{x \geq 2\}\)
+ \([1\times(x \geq 1)] \times \text{wp}(x:= x + 2),\{x \geq 2\}\)
prob. choice
= \([x \geq 1] \times ([x-1] \geq 2) + [x \geq 3] \times ([x+2] \geq 2)\)
assignment
= \([x \geq 1] \times [x \geq 3] + [x < 1] \times [x \geq 0]\)
arithmetic
= \([x \geq 1] \times [x \geq 3] \circ [x < 1] \times [x \geq 0]\)
embedding
= \([x \geq 3] \circ 0 \leq x < 1]\).
logic

For a standard conditional, the reasoning is just “as usual”.

Probabilistic programs: sequential composition

\[ \text{prog}_1;\; \text{prog}_2 \] Execute the first program; then execute the second.

\[
\begin{align*}
\wp(\text{prog}_1;\; \text{prog}_2).B &= \wp.\text{prog}_1.(\wp.\text{prog}_2.B) \\
\end{align*}
\]

\[
\begin{align*}
\wp(\text{c}:= H_{1/2} \oplus T;\; \text{d}:= H_{1/2} \oplus T).[\text{c}=\text{d}] \\
&= \wp(\text{c}:= H_{1/2} \oplus T).\left( \wp(\text{d}:= H_{1/2} \oplus T).[\text{c}=\text{d}] \right) \quad \text{definition} \\
&= \wp(\text{c}:= H_{1/2} \oplus T).\left( \frac{1}{2} \times [\text{c}=\text{H}] + \frac{1}{2} \times [\text{c}=\text{T}] \right) \quad \text{prob. choice; assignment} \\
&= \frac{1}{2} \times (\frac{1}{2} \times [\text{H}]=\text{H}) + \frac{1}{2} \times [\text{H}=\text{T}] \\
&\quad + \frac{1}{2} \times (\frac{1}{2} \times [\text{T}]=\text{H}) + \frac{1}{2} \times [\text{T}=\text{T}] \quad \text{prob. choice; assignment} \\
&= \frac{1}{4} + \frac{1}{4} \quad \text{embedding} \\
&= \frac{1}{2} \quad \text{arithmetic}
\end{align*}
\]

Probabilistic programs: layout of calculations

\[
\begin{align*}
\text{wp.(prog}_1;\; \text{prog}_2;\; \text{prog}_3 \ldots).B \\
\end{align*}
\]

The component programs might have to be written and rewritten many times...

\[
\begin{align*}
\text{wp.(c}:= H_{1/2} \oplus T;\; \text{d}:= H_{1/2} \oplus T).[\text{c}=\text{d}] \\
&\equiv \frac{1}{2} \times [\text{c}]=\text{H} + \frac{1}{2} \times [\text{c}=\text{T}] \quad \text{work from back to front, writing each program fragment only when it is used in the calculation.} \\
\end{align*}
\]

Probabilistic programs: “proper” probabilistic postconditions

\[ wp.(c:= H \times T).(1/2 \times [c=H] + 1/2 \times [c=T]) \]

\[ = \quad 1/2 \times (1/2 \times [H=H] + 1/2 \times [H=T]) \]
\[ + \quad 1/2 \times (1/2 \times [T=H] + 1/2 \times [T=T]) \]
\[ = \quad 1/2 . \]

The expected value of the function \( 1/2 \times [c=H] + 1/2 \times [c=T] \) over the distribution of states produced by the program is \( 1/2 \).

As a special case (from elementary probability theory) we know that the expected value of the function \([\text{pred}]\), for some Boolean \( \text{pred} \), is just the probability that \( \text{pred} \) holds.

That’s why \( wp.\text{prog.}[\text{pred}] \) gives the probability that \( \text{pred} \) is achieved by \( \text{prog} \). But, as we see above, we can be much more general if we wish.

Probabilistic programs: proper post-expectations

The expression \( wp.\text{prog}.B \) gives, as a function of the initial state, the expected value of the “post-expectation” \( B \) over the distribution of final states that \( \text{prog} \) will produce from there.

We call it the greatest pre-expectation of \( \text{prog} \) with respect to \( B \). When \( \text{prog} \) and \( B \) are standard (i.e. non-probabilistic), it is the same as the weakest precondition... except that it is 0/1-valued rather than Boolean.

As a “hybrid”, we have that \( wp.\text{prog.}[\text{pred}] \) is the probability that \( \text{pred} \) will be achieved.

Predicate \( \text{pred} \) holds with probability \( p \), say.

Expectation \([\text{pred}]\) is 1 on \( \text{pred} \) and 0 elsewhere.

Probabilistic programs: demonic choice

\( \text{prog}_1 \triangleleft \text{prog}_2 \) Execute the left-hand side — or maybe execute the right-hand side. Whatever...

\[ wp.(\text{prog}_1 \triangleleft \text{prog}_2).B \triangleq wp.\text{prog}_1.B \min wp.\text{prog}_2.B \]

In the case of recursion, however, we cannot give a purely syntactic definition. Instead we say that

\[ (\text{mu } \mathbf{z} z \cdot C) := \text{least fixed-point of the function } \text{cntx} : \mathbb{T}S \to \mathbb{T}S \]

\[ \text{defined so that } \text{cntx}(wp.\mathbf{z}z) = wp.C. \]

Figure 1.5.3. Probabilistic wp-semantics of \( \mathbb{pGCL} \)
Exercises

Ex. 1: Probabilistic then demonic choice

Calculate wp. \((c := H^{1/2} \oplus T; \ d := H \cap T )\). \([c=d]\).

Ex. 2: Demonic then probabilistic choice

Calculate wp. \((d := H \cap T; \ c := H^{1/2} \oplus T )\). \([c=d]\).

Ex. 3: Explain the difference

The answers you get to Ex. 1 and Ex. 2 should differ. Explain “in layman’s terms” why they do.

(Hint: Imagine an experiment with two people and two coins, in each case.)

Ex. 4: The nature of demonic choice

It is sometimes suggested that demonic choice can be regarded as an arbitrary but unpredictable probabilistic choice; this would simplify matters because there would then only be one kind of choice to deal with.

Use our logic to investigate this suggestion; in particular, look at the behaviour of

\(c := H^{1/2} \oplus T; \ d := H_p \oplus T\) for arbitrary \(p\),

and compare it with the program of Ex. 1. Explain your conclusions in layman’s terms.

Ex. 5: Compositionality

Recall programs

\(A: \) \( coin := edge \cap (coin := heads^{1/2} \oplus coin := tails) \)

\(B: \) \( (coin := edge \cap coin := heads^{1/2})\oplus (coin := edge \cap coin := tails) \),

which we now call \(A\) and \(B\). Say that they are similar because from any initial state they have the same worst-case probability of achieving any given postcondition. (We showed this by tabulation.)

Find a program \(C\) such that \(A;C\) and \(B;C\) are not similar (even though \(A\) and \(B\) are). (Use the \(wp\)-definition of “;”.) What does this tell you about the simple program logic we considered briefly at the beginning?

More generally, let \(A\) and \(B\) be any two programs that are similar, but not equal in our \(wp\) logic. Show that there is always a program \(C\) as above, \(i.e.\) such that \(A;C\) and \(B;C\) are not similar. What does that tell you about our quantitative logic when compared to the simple logic?